Panda Behaviour for Unity
User Manual

www.PandaBehaviour.com

Copyright 2015 Eric Begue.
Please send comments/corrections/feedbacks to ericbeg+panda@gmail.com

Table of Contents

Overview....3
Installation....3
Panda Behaviour (Script) Component....4
  Component properties....4
  Node Folding....5
  Passive Mode....5
  Live View Mode....5
  Debugging....6
    Break Points....6
    Debug Info....6
    Built-in tasks....6
Implementing task in C#....6
  Return type and value....6
  Task parameters....6
  Task.current....7
  Examples....7
Panda Script Syntax....7
  Node types....7
  Node parameters....7
  Indentation....7
  Structural nodes....8
    BEHAVIOUR....8
    SEQUENCE....8
    FALLBACK....9
    PARALLEL....9
    RACE....10
    REPEAT....10
    WHILE....11
    IF....12
    NOT....12
    MUTE....13

Overview

Panda Behaviour is a Behaviour Tree (BT) engine integrated with a simple scripting language named Panda Script. You have now the the power of BT at your finger tips. Write high-level and scalable AI logic with the flexibility and simplicity of plain text script. Though Behaviour tree is a tool for programmers, its simplicity requires only basic knowledge about programming to start using it.

Panda Behaviour is build in order to easily iterate over the design of your game BT logic: make modifications to your scripts, refresh Unity and see the result of your changes in play mode without waiting for any compilation. Furthermore, you can visualize the runtime execution of your BT and debug it in realtime.

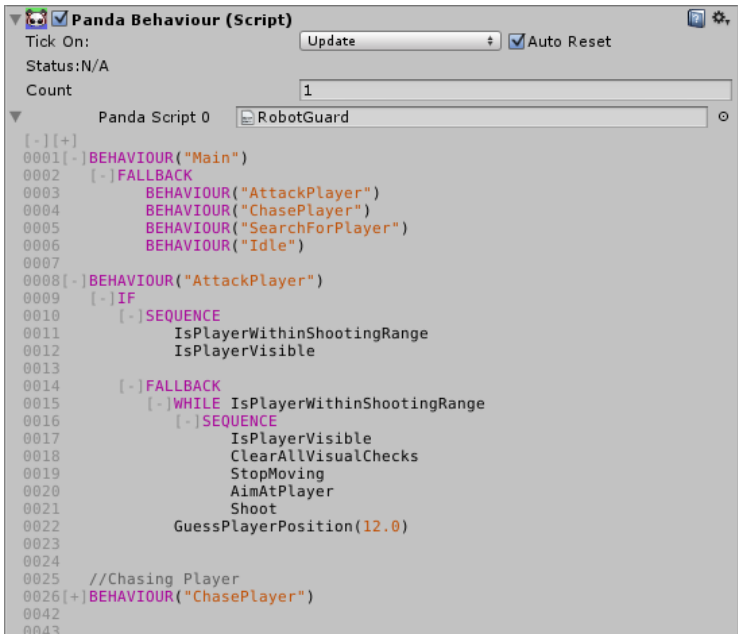
To keep things simple, Panda Behaviour features scope is kept to the bare minimum: a BT engine. You know what you need for your game, and what BT tasks you need. Creating a custom BT task is as simple as writing a c# sharp method, in fact this is the way you define a new task!

Please have a look at the provide examples in PandaBehaviour/Examples to get started.

Installation

Download the Panda Behaviour package from the Unity Asset Store, then import it into your project. Once the package has been imported, a new directory named PandaBehaviour will be present at the root of your the Assets folder. The PandaBehaviour folder contains the core of Panda Behaviour, the documentation and some examples to get started.

Panda Behaviour (Script) Component



The Panda Behaviour component is the only interface you have to deal with within Unity. This component determines the Behaviours Tree of the Game Object it is attached to. The Panda Behaviour component intregrate a syntax highlighted script viewer in two mode: Static View and Live View for respectively inspecting the code and visualize the behaviour tree at runtime.

To be functionnal, a Panda Behaviour requires one or more Panda scripts to be assigned to it. A Panda script is a simple TextAsset in your project wich contain a script describing a BT which you can edit using your favorite text editor (You can create a new Panda Script asset from the context menu of the project view: right click on the project view tab > create > Panda Script). Each task mentionned in the scripts has to be implemented by a unique c# sharp method present in any component attached to the Game Object. It is also required that any BEHAVIOUR node is defined within the attached Panda Scripts.

Once the functionnal requirements of a Panda Behaviour are full-filled, every task will be bound to their respective c# sharp method based on the task name and parameter types. At this point the Panda Behaviour is ready to be used.

A Panda Behaviour is evaluated each time the Tick method of the Panda Behaviour component is called, which as for effect to run the scripted behaviour tree and called the c# sharp methods accordingly.

During the execution of Panda Behaviour, a live view of the execution is available in the inspector, which provides a useful insight about the current state of the behaviour tree, as well as it's execution flow and other debugging information.

Component properties

Property	Funtion
Tick On	Determines when to tick the root tree. The options are: Update, LateUpdate, FixedUpdate or Manual. When the manual option is choosen, the PandaBehaviour component will not tick the root tree automatically. This is useful if you want to execute the behaviour tree at a custom time by calling PandaBehaviour.Tick() from your own script.
Auto Reset	When enable, if the tree is ticked and has previously returned Succeed or Failed the tree will be automatically reset. When AutoReset is disable, the tree will be run once and calling Tick() on it will has no effect.
Status	Displays the current status of the root node. N/A indicates that the behaviour tree is not active.
Count	The number of Panda scripts defining the behaviour tree

Node Folding

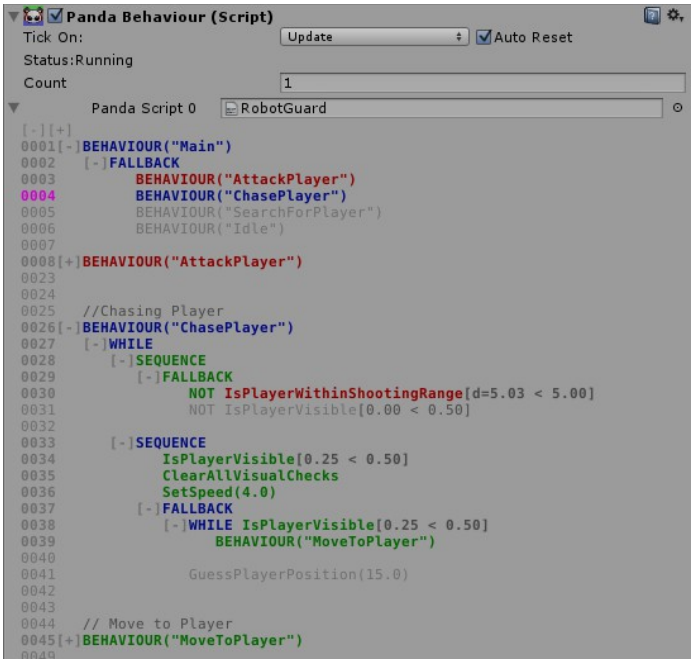
You can use the [–] and [+] buttons to respectively collapse and unfold the source code hierarchy.

Passive Mode

When the behaviour is not active, the syntax highlighting is in Passive Mode. The color indicates the element type:

- Purple: structural nodes
- Orange: node parameters
- Black/White (Light/Dark skin) : Tasks
- Grey: comments

Live View Mode



When the behaviour is executing, the syntax highlighing changes to the Live View mode. In this mode, each node is colored according to its current status:

- Green: The node has Succeeded.
- Red: The node has Failed.
- Blue: The node is Running.
- Light grey: The node is Ready (it has not been ticked).
- Dark grey: Comments or debug info

A bold font indicates that the node has been ticked on the current frame.

Debugging

Break Points

You can set break points by clicking a line number of a line containing nodes. The line number will turn yellow indicating that the break point is set. When the execution flow reaches a node on a break point, the game will be paused and the line number will turn purple. You can resume the game by clicking on the break point again, or using the usual pause or next frame button.

Note that the break point will be activated only when a node has been ticked for the first time, that is the node status was ready (grey) before it was ticked.

Debug Info

You can display useful information that would give insight into how a task is currenty performing.

In the life view, Debug Info is visible in the inspector next to a node. You can set the displayed text from the c# implementation of the task by setting the followin string:

Panda.Task.current.debugInfo

Built-in tasks

The Panda Behaviour component come with a minimal set of built-in tasks:

Debug tasks wrapper

These tasks wrap the Unity debug utilities:

- DebugLog("*message*"): Debug.Log(string) wrapper.
- DebugWarning("*message*"): Debug.Log(string) wrapper
- DebugError("*message*"): Debug.Log(string) wrapper
- DebugBreak: Debug.Break wrapper

Wait

- Wait(*duration*), this task returns Succeeded until *duration* is elapsed.

Implementing task in C#

Every task in a Panda Script requires a C# implementation present on a MonoBehaviour on the GameObject. A task is implemented as a method returning a Panda.Status and that as the exact same name as the task in the script.

Return type and value

A method implementing a task must return a Panda.Status and the return value must be one of the following be :

- Status.Succeed : Indicates that the task has succesfully complemented
- Status.Failed: Indicates that the taks is complemented but has failed.
- Statis.Running: Indicates that the tasks is still running and requires more Ticks (or frames) to be completed.

Task parameters

A task might take none or several parameters which are written after the taskname within paramethesis and separated by comas. Parameters are only values (no variable identifiers). For each task, the C# implementation is expected to take the same number and type of parameters. It's possible to have task overloading (tasks with same name but with different number of parameters or parameter type).

Panda Script supports the following value type:

- bool
- int
- float
- string

Task.current

Within the scope of a task method, you can have access to the runtime Task object through Panda.Task.current, which provides information about the the current task. Usually, you only need to access the following members from a task method:

status

- status: the current task status (alternatively, you can use the bool shortcuts: isStarting, isRunning, hasFailed, hasSucceeded)
- debugInfo, a string visible next to a node in the inspector to display debugging information.

Examples

Let's we want to implemented the followring behaviour tree:

```
BEHAVIOUR("Main")
      Print("Hello World!")
```

This Panda Script contains a task name 'Print' taking a string as parameter. The Panda Behaviour Tree engine expects that one and only one method with the same name and signature exist on any component attached to the Game Object. Here we define a new component named “HelloWorld” that implements the task “Print”:

```
using UnityEngine;
using Panda;

public class HelloWorld : MonoBehaviour
{
      Status Print(string message)
      {
            Debug.Log(message);
            return Status.Succeeded;
      }
}
```

Panda Script Syntax

Panda Script is a simple Behaviour Tree language. Panda Scripts are written into plain text file, they are TextAsset into a unity project. This section describes the Panda Script syntax.

Node types

There are two types of node: structural node and task node:

- Structural nodes defines the tree hierarchy and the execution flow of the behaviour tree.
- Task nodes are leaf node that execute low level operation implemented as a c# method.

Node parameters

A node can have parameters. Parameters is a list of coma separated values enlosed into parathesis after a node name. Panda script defines 4 parameters types: string, int, float, bool. The parameters are passed to the c# method

implementing the task when the task is ticked.

Indentation

In a Panda Script file, indentations are used to define the hierarchy of the tree. A child node has a higher indentation than it's parent. Both tabulations and spaces are supported as white space to indent the code. For consistency, though a mix of spaces and tabulations would work, it is recommended to use the same type of indention in the same file.

Structural nodes

Structural nodes define the hierarchy and the execution flow of the behaviour tree. Each structural node have a specific algorithm to tick its child and to the return its status. This section describe all the 10 structural nodes, their syntax, how their children are ticked and in which situation they return Succeeded, Failed or Running.

The information contains is this section is also available into a one page table in the file PandaScript.quickref.pdf.

BEHAVIOUR

Syntax

```
BEHAVIOUR("name")
[child]
```

Description

- A tree definition when rooted.
- A sub tree when parented to a node.

Returned status

Returns **Succeeded** when child returns **Succeeded**.
Returns **Failed** when child returns **Failed**.
Returns **Running** when child returns **Running**.

On Tick

Ticks *child* and returns status.

SEQUENCE

Syntax

```
SEQUENCE
      child_0
      ...
      child_i
      ...
      child_k
```

Description

A composite node that runs all children from top to bottom unless one **Failed**.

Returned status

Returns **Succeeded** when all child **Succeeded**.
Returns **Failed** when one child **Failed**.
Returns **Running** when one child is **Running**.

On Tick

Ticks each child from top to bottom and returns status. Returns status on the first child that **Failed** or that is **Running**.

FALLBACK

Syntax

```
FALLBACK
      child_0
      ...
      child_i
      ...
      child_k
```

Description

A composite node that runs all children from top to bottom until one **Succeeded**.

Returned status

Returns **Succeeded** when one child **Succeeded**.
Returns **Failed** when all child **Failed**.
Returns **Running** when one child is **Running**.

On Tick

Ticks each child from top to bottom and returns status. Returns status on the first child that **succeed** or that is **Running**.

PARALLEL

Syntax

```
PARALLEL
      child_0
      ...
      child_i
      ...
      child_k
```

Description

A composite node that runs all children on the same tick until one child **Failed**.

Returned status

Returns **Succeeded** when all child **Succeeded**.
Returns **Failed** when one or more child **Failed**.
Returns **Running** when one or more child is **Running** and none has Failed.

On Tick

Ticks all child from top to bottom, then returns status.

RACE

Syntax

```
RACE
      child_0
      …
      child_i
      …
      child_k
```

Description

A composite node that runs all children on the same tick until one child **Succeeded**.

Returned status

Returns **Succeeded** when One or more child **Succeeded**.
Returns **Failed** when All children **Failed**.
Returns **Running** when One or more child is **Running** and none has Succeeded.

On Tick

Ticks all child from top to bottom, then returns status.

REPEAT

Syntax

```
REPEAT[(n)]
      child
```

Description

A repeater node that repeat Child *n* times unless *child* **Failed**. If *n* is not specified or *n*=0, the repetitions is infinite.

Returned status

Returns **Succeeded** when :

- The child has been repeated *n* time and **Succeeded** at each iteration.
- Never if *n* is not specified or *n*=0 (infinite repetition).

Returns **Failed** when The *child* **Failed**.
Returns **Running** when The number of iterations has not reached *n* and the child did not **Failed**.

On Tick

Ticks child if number of iteration has not reached *n* and returns status.

WHILE

Syntax

```
WHILE [condition]
      child
```

Description

A repeater node that repeats Child unless condition **Failed** or Child **Failed**.

If condition is not specified, the repetitions is infinite.

Returned status

Never returns **Succeeded**,

Returns **Failed** when:

- *condition* **Failed**.
- *condition* **succeed** and child **Failed**.

Returns **Running** when *condition* **succeed** and either:

- *child* **Succeeded**.
- *child* is **Running**.

On Tick

Ticks *condition, if condition:*

- **Succeeded**, ticks child then returns status.
- **Failed**, returns status

IF

Syntax

```
IF condition
      then-child
      [else-child]
```

Description

A conditional node to implement an if-then-else branching.

Returned status

Returns **Succeeded** when:

- *condition* **Succeeded**
  and *then-child* **Succeeded**.
- *condition* **Failed**
  and *else-child* **Succeeded**.

Returns **Failed** when:

- *condition* **Failed**.
- *condition* **Succeeded**
  and *then-child* **Failed**.
- *condition* **Failed**
  and *else-child* **Failed**.

Returns **Running** when:

- *condition* is **Running**.
- *condition* **Succeeded** and *then-child* is **Running**.
- *condition* **Failed** and *else-child* is **Running**.

On Tick

Ticks *condition,* if *condition*:

- **succeed**, ticks t*hen-child* and returns status.
- **Failed**, ticks *else-child*, if exists, and returns status.

NOT

Syntax

```
NOT child
```

Description

An inverter node. Returns **suceeded** when its child **Failed** and returns **Failed** when its child **Succeeded**.

Returned status

Returns **Succeeded** when the *child* **Failed**.
Returns **Failed** when the *child* **Succeeded**.
Returns **Running** when the *child* is **Running**.

On Tick

Ticks child and returns status.

MUTE

Syntax

```
MUTE child
```

Description

A muter node, ignores *child* status. Returns **Succeeded** whenever its *child* returned **Succeeded** or **Failed**.

Returned status

Returns **Succeeded** when the *child* either **succeed** or **Failed**.
Never returns **Failed**.
Returns **Running** when the *child* is **Running**.

On Tick

Ticks *child* and returns status.